State of infrastructure drift 2021

What we learned asking 200 DevOps teams about infrastructure drift

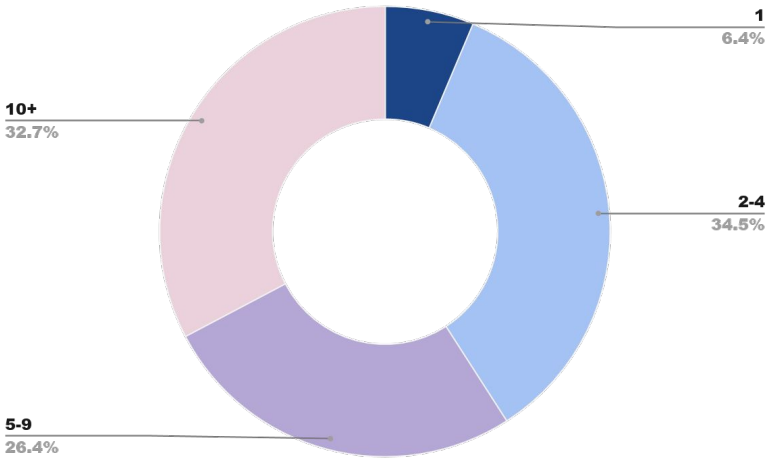
Table of contents

Introduction

At Cloudskiff, we are working on protecting codified cloud infrastructures.

We spoke to 200 infrastructure teams, to see what issues they were facing. Keeping those infrastructures in sync and avoiding drift is a real challenge for them. Indeed, drift has consequences on toil and efficiency, forces teams to put in place strict controls that decrease flexibility, and can have a security impact.

In this study, we will describe our findings and a few options to tackle drift. We interviewed 50+ teams to collect stories and feedback, and surveyed 200 teams of all size.



How many people in your organization are actively building / maintaining infrastructure?

Unexpected learnings

- **Application and deployment induced drift** is a widely spread nuisance.
- **Security issues** count among major concerns.
- **GitOps only can’t screen out drift.**

Key takeaways

Drilling down

50% of teams are subjected to infrastructure drift due to unintentional / non manual changes

For a vast majority of teams (96%) the main cause of drift is when a team member makes a manual change through the Cloud provider's (AWS, Azure, etc) console. But **50% of teams also record drifts due to uncontrolled changes**, be it on the applications' side, through the deployment stack and/or on the cloud providers' side.

Security issues count among the most important impacts of drift

Drift has costly impacts for most teams. It has consequences on toil and efficiency and forces teams to put in place strict controls that decrease flexibility. **Even more noticeable is that close to 20% of teams consider security issues the most important impact of infrastructure drift.**

Each solution comes with its own limitations

There is no perfect solution to deal with drift. **Even a full GitOps workflow** with restricted access to environments **comes with a tradeoff** in terms of visibility, flexibility or security and **cannot prevent non manual drift.**

What’s drift?

First, let us agree on a definition. Drift is a multi-faceted problem, but most of the people we talked to agreed on the following definition:

“Infrastructure drift is when there is an unwanted delta between the IaC code base and the actual state of the infrastructure.”

This issue becomes more and more complex as the number of environments grows. Some teams have dozens of environments that they need to keep updated.

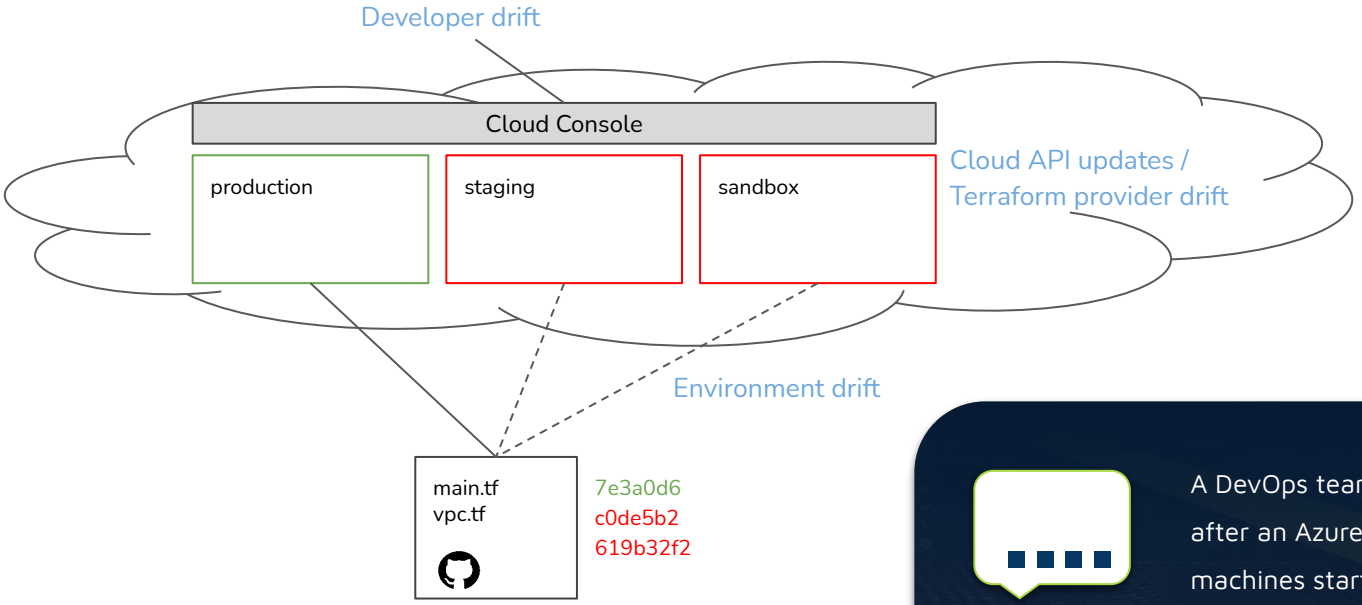
We identified **three main causes of drift**:

- 96% of teams: a team member makes a change through the (AWS, Azure, etc) console or directly updates infrastructure resources through an application API.
- 44% of teams: a team member applies an IaC change to an environment but does not propagate it to other environments.
- **50% of teams: application and deployment induced drift.**

While the first two are mostly workflow issues (we will come back to how some teams adjusted their workflow to reduce such behaviors, with more or less success), **the last one comes as a big surprise as regards its volume. We knew unintentional application and deployment induced drift was a reality, but didn’t expect such a massive spread.**

It is interesting to note though, that the third cause is completely independent of the DevOps team.

While unpredictable, this kind of change can cause massive headaches.



A DevOps team lead told us this story: after an Azure API update, all his machines started denying access.

It took a while to find out that Azure's API and Terraform provider had been updated, and a new "identity" parameter was now required to grant access.

We heard testimonials of teams who experienced **drift linked to an application.** The behaviour of this kind of application is sometimes poorly monitored, and can generate complex drift scenarios. In one interview, **an internal app was updating EBS parameters. Terraform plan does not show this, generating a false sense of control.**

The truth that comes back about drift is that once you think you have it under control, you don't.

One SRE manager told us that he had Terraform-ed his datadog monitoring, and thought everything was under control there. Developers have access to Datadog - what's the point of monitoring if they don't.

A developer in the team thought that increasing the frequency of the Datadog probe to once per minute would be cool, which is easy to do through the GUI. That change remained unseen for a while and generated huge costs, until the next datadog bill.

The consequences of drift

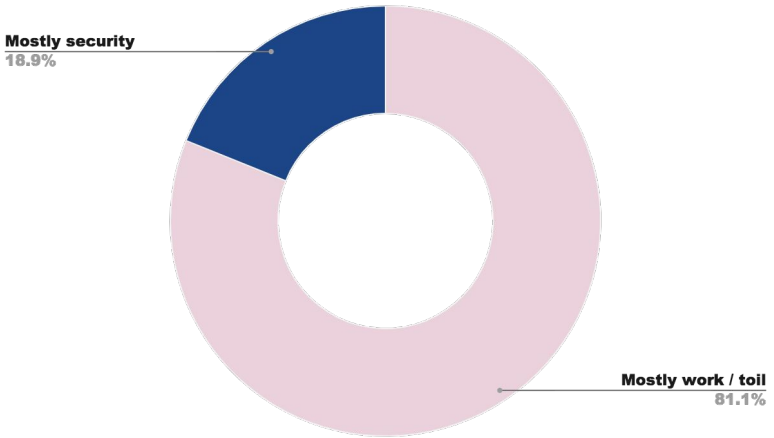
Beyond toil, security issues also rank among the most important impacts of drift

Why does infrastructure drift matter?

Everybody seems to agree that drift is annoying, but we tried to quantify its impact.

Drift mostly causes additional work, as well as security issues. In one of our interviews, a DevOps lead analyzed the problem quite clearly: **every drift event causes uncertainty, a resolution time, and a potential security issue.**

Even more noticeable is that close to 20% of teams consider security issues the most important impact of infrastructure drift.

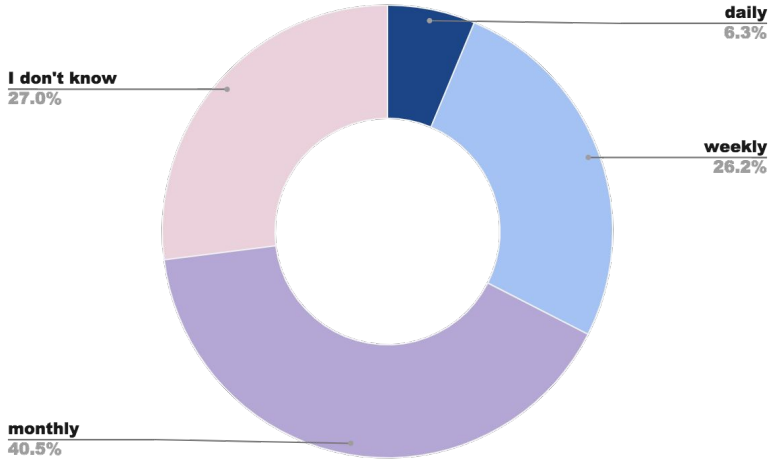


What is the impact of a drift event?

There are also more subtle effects. To avoid excessive drift, some teams make significant adjustments to their workflows. In some teams, only the team lead is allowed access to production. In others where developers are not skilled at IaC, getting a small change to environments done goes through a painful and long ticketing system. In other words, drift causes issues, which leads to rigid / counterproductive processes, which leads to a **decrease in speed and flexibility.**

When asked how often they have a drift event, roughly one-third of teams face drift at least weekly, one-third monthly.

The last third is interesting: **some teams do not know if and when drift happens but when prompted all of them can relate a recent drift event**. In those teams, drift is a visibility issue: it is not tracked, or measured, or observed, thus it cannot be kept under control or improved.



How often do you have a drift event?

We noticed that teams building new products and infrastructure tend to face a lot of drift in the early development stage, because team members experiment, change things through the console, and don't follow processes.

As time goes, and the infrastructure matures, drift tends to happen less frequently and change in nature.

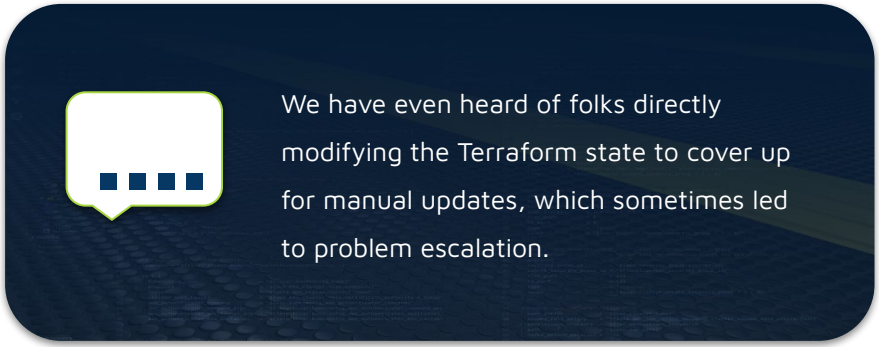
But it does not mean the challenge is solved.

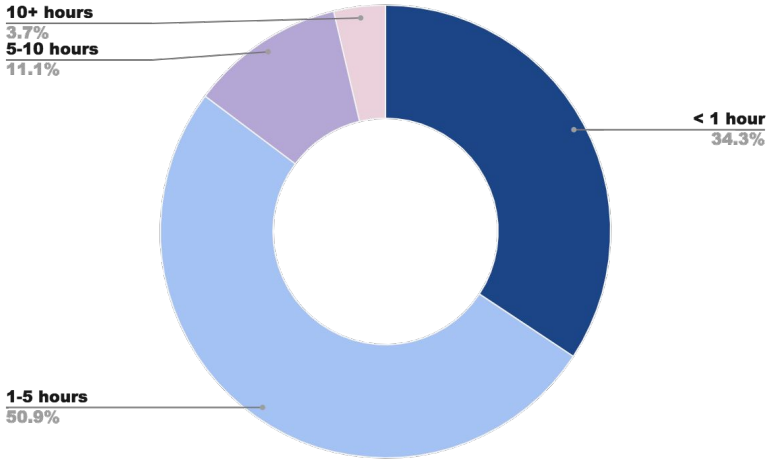
We heard stories from several service companies that operate with a build & run model. A team of DevOps builds the infrastructure for a customer. They tend to have good Terraform skills and follow best practices such as work in CI/CD.

Then a run team of system administrators takes over the operation of the infrastructure. They tend to be only casual Terraform users, and use the console to make changes. This causes a lot of drift and manual intervention to fix issues.

We also polled DevOps about how long it took to solve a drift event. **Beyond the time spent, drift causes context switches and losses of productivity. Important updates can be delayed because the team lead is busy tracing back who made a change through the console, and whether it is OK to override it with an apply or not**.

Sometimes the author of the change themselves doesn't remember what the change was for.





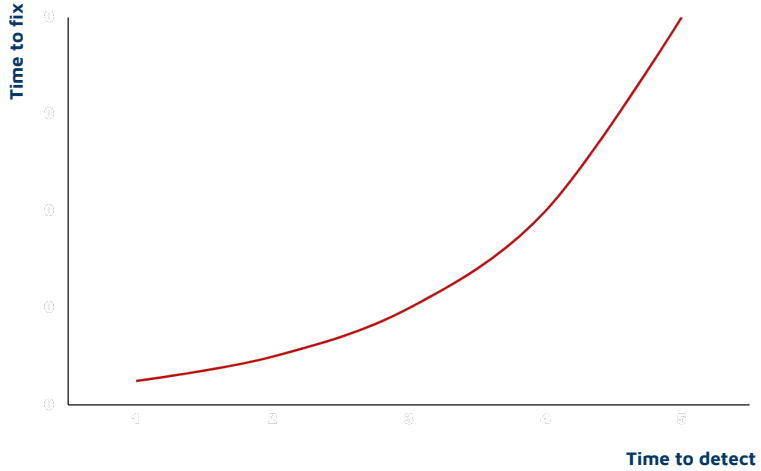
How much work does a drift event take to resolve, on average (hours)?

Current solutions

(illustration)

Most teams (60%) identified that to tackle drift, the first thing they needed to do was **restrict access to production** to a few team members. That does not solve the problem of staging environments drift, as only 25% of teams restrict access to staging environments: doing so requires being capable of and paying for dedicated sandbox environments for developers.

Beyond that, some teams (30%) **implement policies** (using OPA, or cloud-provider specific permission systems) to define acceptable policies and behaviour. This prevents some of the drift, but not all of it.



What we found in our conversations was that drift always happens, and the key challenge is being able to detect and analyze it in that case. The faster it is detected, the easier it is to remediate drift, which is why 30% of the DevOps we interviewed had a `terraform plan` in a cron job.

Solutions vs impacts in a nutshell

	Full GitOps workflow	terraform plan in a CRON job	Restrict access to the production environment	Restrict access to staging environments
Prevents developer generated drift	yes	no	partially	yes
Prevents cloud-provider generated drift	no	no	no	no
Makes drift visible	no	partially	no	no
Analyzes drift root cause	no	no	no	no
Limitations	Hard to rollout in legacy / complex environments.	Terraform plan does not “see” some changes		Decreases developer speed or requires the cost and capability of deploying sandbox environments

Limitations

Analyzing what solutions are deployed against drift in 200 teams led us to discover how poorly the topic is addressed.

- **Deploying a full GitOps workflow is a good option in theory, but hard to do in practice** for most teams as it strictly limits access and thus lacks flexibility.
- **Even with a full GitOps workflow, cloud provider updates**, or any unintentional changes done to the infrastructure through the cloud **API cause drift.**
- **Running terraform plan** (or the equivalent) **catches some drift, but not all**. Security groups are the best example of this. Opening up a security group to all traffic through the console remains undetected in a Terraform plan, though it is a critical security event.
- **Restricting access to environments to prevent drift is a good option, but it comes with a tradeoff**: developers lose flexibility. It also requires that the team has enough Terraform bandwidth to effectively meet demands for infrastructure update.

About driftctl

Take control of infrastructure drift

Infrastructure as code is awesome, but there are so many moving parts: codebase, `*.tfstate`, actual cloud resources. Things tend to drift.

driftctl is a free and open-source CLI that tracks, analyzes, prioritizes, and warns of infrastructure drift

More info at www.driftctl.com

MIND THE GAP
between the code and the platform